

SANDIA REPORT

SAND2015-3275

Unlimited Release

Printed September, 2015

Reducing Communication Costs for Sparse Matrix Multiplication within Algebraic Multigrid

Grey Ballard, Jonathan Hu, Christopher Siefert

Prepared by

Sandia National Laboratories

Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



Reducing Communication Costs for Sparse Matrix Multiplication within Algebraic Multigrid

Grey Ballard, Jonathan Hu, and Christopher Siefert

Abstract

We consider the sequence of sparse matrix-matrix multiplications performed during the setup phase of algebraic multigrid. In particular, we show that the most commonly used parallel algorithm is often not the most communication-efficient one for all of the matrix-matrix multiplications involved. By using an alternative algorithm, we show that the communication costs are reduced (in theory and practice), and we demonstrate the performance benefit for both model (structured) and more realistic unstructured problems on large-scale distributed-memory parallel systems. Our theoretical analysis shows that we can reduce communication by a factor of up to 5.4 for a model problem, and we observe in our empirical evaluation communication reductions of factors up to 4.7 for structured problems and 3.7 for unstructured problems. These reductions in communication translate to run-time speedups of up to factors of 2.3 and 2.5, respectively.

1 Introduction

Algebraic multigrid (AMG) is an efficient method for solving a large sparse linear system $Ax = b$ arising from a self-adjoint elliptic partial differential equation (PDE). The method involves, during a setup phase, generating a sequence of related systems of decreasing size and then, during a solve phase, using all systems to iteratively improve the solution to the original system. The sequence of systems is generated algebraically; that is, the systems are constructed from A without geometric knowledge of the underlying PDE.

On large-scale distributed-memory parallel machines, the computation time of the setup phase is dominated by a sequence of sparse matrix-matrix multiplication (SpMMs) involving matrices distributed across processors. In this paper, we show that the most commonly used parallel SpMM algorithm is often not the most communication-efficient one for all of the matrix multiplications involved. By using an alternative algorithm, we show that the communication costs are reduced (in theory and practice), and we demonstrate the performance benefit for both model and real problems on large-scale parallel systems.

We consider the smoothed aggregation multigrid method, described in more detail in Section 3. After fine-level rows of A are grouped into coarse-level aggregates, there are three sparse matrix multiplication operations performed to construct the coarse grid operator. The tentative prolongation matrix \hat{P} represents aggregate membership, and the final prolongation matrix P is computed by applying a step of damped Jacobi to \hat{P} , has the same sparsity structure as the product $A \cdot \hat{P}$. In the symmetric case, the coarse grid operator is given by the triple product $A_c = P^T A P$, which is typically performed with two more sparse matrix multiplications: $A \cdot P$ and $P^T \cdot (AP)$.

The standard approach for performing each of these parallel sparse matrix multiplications is to use a row-wise algorithm: for general $C = A \cdot B$, each processor owns a subset of the rows of A , a subset of the rows of B , and computes a subset of rows of C (which matches the distribution of A). The communication required is an exchange of rows of B among processors. We consider as an alternative the outer-product sparse matrix multiplication algorithm, where each processor owns a subset of the rows of A , the corresponding subset of *columns* of B , and the communication consists of exchanging partially unreduced rows of C among processors (assuming the desired output distribution is row-wise). We describe these general algorithms in more detail in Section 4.

Because of its low arithmetic intensity, the performance of parallel algorithms for sparse matrix multiplication is typically bound by the cost of interprocessor communication and data movement within each processor's memory hierarchy. Based on interprocessor communication cost analysis of model problems, we conclude that the row-wise algorithm is the most communication-efficient choice for computing $A \cdot \hat{P}$ and $A \cdot P$ but the outer-product is the better choice for computing $P^T \cdot (AP)$. For a fine grid operator corresponding to a 3D 27-point stencil on a regular grid, for example, the row-wise algorithm for $P^T(AP)$ requires more than $5\times$ the communication of the outer-product algorithm. Furthermore, using the outer-product algorithm for $P^T \cdot (AP)$ avoids the explicit redistribution of P^T that is typ-

ically required in order to use the row-wise algorithm and involves as much interprocessor communication as a sparse matrix multiplication. We provide the theoretical analysis for these conclusions in Section 5.

We have implemented the outer-product algorithm within the Trilinos software framework, and we compare communication costs and running times of the two approaches for representative matrices. Trilinos is a Sandia-centered collection of high performance numerical libraries that target current and next-generation parallel computer architectures. Of interest for the current discussion are the Tpetra and MueLu libraries. Tpetra provides distributed sparse linear algebra services for maps, multivectors, and matrices [2]. These form the basis for many Trilinos linear solver and preconditioner libraries. In particular, the Trilinos multigrid library MueLu [22] depends on Tpetra. MueLu provides a variety of aggregation-based linear multigrid preconditioning algorithms. Solvers based on the Tpetra stack, including MueLu, have been demonstrated to provide effective, scalable preconditioners for large-scale parallel fluid applications [20]. For a more complete overview of Trilinos, please see [19].

To confirm the theoretical analysis for sparse matrix multiplication within algebraic multigrid, our experiments include regular-grid stencil matrices for 2D and 3D problems. We also perform tests in realistic settings, arising from a low Mach fluid dynamics application problem involving unstructured grids to demonstrate the benefits of the new approach. These experimental results are presented in Section 6. For the model problems, we observe reductions in the amount of data communicated of up to $4.7\times$ and run time improvements of up to $2.3\times$. For the unstructured problems, we observe reductions in communication of up to $3.7\times$ and speedups of up to $2.5\times$.

To summarize, the main contributions of this work include

- a scalable implementation of the outer-product algorithm in the Trilinos framework;
- theoretical communication cost analysis of various SpMM approaches for the setup of a model AMG problem;
- experimental validation of reductions in communication and run time for both model and application problems; and
- an argument that both row-wise and outer-product algorithms for SpMM have an important role to play during AMG setup.

2 Related Work

Many previous works have considered distributed-memory algorithms for sparse matrix-matrix multiplication, both for general-purpose use [1, 4, 10] and for specific applications [7, 12]. Buluç and Gilbert [10] propose, analyze, and evaluate the Sparse SUMMA algorithm. This algorithm is based on the SUMMA algorithm for dense matrices [25] and has a communication pattern that ignores the sparsity structure of the matrices; random permutations of rows and columns of the input matrices encourages load balance across processors. Borštnik et al. [7] use a similar idea for their proposed parallel algorithm, converting another dense algorithm (Cannon’s [11]) to the sparse case and relying on random permutation to achieve load balance. This work focuses on quantum-chemical applications and involves optimizations particular to the application, including tuning local computations for dense subblocks of various sizes and filtering small entries. Challacombe [12] proposes using the row-wise algorithm (presented in Section 4.1) for another application from quantum chemistry. Akbudak and Aykanat [1] consider the outer-product algorithm (presented in Section 4.3) for matrices arising in several application areas. They propose a hypergraph model to represent the communication costs particular to input sparsity structures and use hypergraph partitioning software to choose the best data-partitioning scheme. Ballard et al. [4] consider multiple algorithms, classifying them into 1D (described in Section 4), 2D (which include Sparse SUMMA and Sparse Cannon), and 3D varieties. They analyze and compare the communication costs for multiplication of Erdős-Rényi random matrices and also prove expectation-based communication lower bounds for those inputs.

Other works have addressed the sparse matrix multiplications specifically occurring within the setup phase of algebraic multigrid, including algorithms and implementations for sequential [21], GPU [6, 13, 16], and distributed-memory parallel [5, 24] platforms. McCourt, Smith, and Zhang [21] use a matrix coloring technique to cast the sparse matrix times sparse matrix operation as a sparse matrix times dense matrix operation, and they show benefits of the technique for matrices coming from geometric-algebraic multigrid with a sequential implementation. Bell, Dalton, and Olsen [6] describe efficient GPU implementations for both the setup and solve phases of algebraic multigrid, including a sparse matrix-matrix multiplication algorithm based on fine-grained parallelism for computing the Galerkin triple product. Further improvements and GPU optimizations for sparse matrix multiplication are described in a subsequent paper [13]. Gremse et al. [16] also consider algebraic multigrid on the GPU and use a technique called “row merging” to reduce the problem to multiplying matrices with simplified structure and performing an efficient row merge operation. Tuminaro and Tong [24] describe smoothed aggregation algebraic multigrid on distributed-memory machines; their approach is implemented within the ML package [15], which uses the row-wise algorithm (presented in Section 4.1) for the sparse matrix-matrix products. Ballard et al. [5] use a hypergraph model to characterize the communication costs of parallelization schemes for general sparse matrix multiplication and consider the Galerkin triple product as a case study; they conclude that the row-wise algorithm is communication efficient for the first of the matrix-matrix products but inefficient for the second.

PETSc [3] provides a variety of options for performing the Galerkin triple-matrix product.

The default option is to use the outer-product algorithm for the $P^T \cdot (AP)$ multiplication, which avoids the redistribution of P^T . The PETSc developers have observed that the row-wise algorithm for $P^T \cdot (AP)$, where P^T is already row-distributed, is significantly faster than the outer-product algorithm. However, including the cost of the redistribution of P^T in the row-wise approach, they have observed that the outer-product approach is more efficient overall [28]. In contrast, Hypre [14] performs the Galerkin triple-matrix product in a single pass, rather than as two separate matrix-matrix multiplies [27].

3 Algebraic Multigrid using Smoothed Aggregation

Algebraic multigrid is a provable scalable solution method for sparse linear systems

$$Ax = b \tag{1}$$

arising from self-adjoint elliptic partial differential equations [8, 9, 17, 23].

In an AMG method, a sequence of linear systems of decreasing size, $A_i x_i = b_i$, are generated and used to accelerate the solution of (1). In this sequence, $i = 0$ corresponds to (1). Associated with each linear system is a solution method called a *smoother*. The smoother is an iterative method such as Gauss-Seidel, a polynomial method, an incomplete factorization, or even a Krylov method. In a properly constructed AMG method, each system in the sequence resolves a particular range of errors that can be quickly reduced by its smoother. On regular meshes, the errors that are rapidly reduced by the smoother are oscillatory in nature. Errors that are smoothly varying are handled by later systems in the sequence. Information is transferred between the sequence's systems with interpolation (*prolongation*) matrices P_i and *restriction* matrices R_i . For symmetric problems, $R_i = P_i^T$, and $A_{i+1} = R_i A_i P_i$ for $i > 0$. In AMG, the main algorithmic challenge is the automatic creation of the P_i 's and R_i 's. For smoothed aggregation, this entails paying special attention to the near nullspace of A_0 , which we refer to as \mathcal{N} . This is usually taken to be the nullspace of the problem without any boundary conditions applied.

We now briefly outline the steps for constructing P_i using a smoothed aggregation multigrid method. More details can be found in [26]. First, coarse level degrees of freedom (DOFs) are created by grouping fine level rows together into *aggregates*. The final prolongator will have N global rows and M global columns, where N is the number of fine level rows and M is the number of aggregates. Second, an intermediate *tentative* prolongator \hat{P} is created. The vectors in the near nullspace \mathcal{N} are rewritten to have local support over aggregates. This means each vector z in \mathcal{N} is expanded to a set of vectors $v_i, 0 < i < M$. For each vector v_i

$$v_i(j) = \begin{cases} z(j), & \text{if DOF } j \in \text{aggregate } i \\ 0, & \text{otherwise} \end{cases} . \tag{2}$$

The v_i 's are collected into a matrix B , which is a block rectangular (tall and skinny) matrix, where each block corresponds to an aggregate. Each aggregate block in B is orthonormalized via a local QR decomposition. The resulting orthonormal factors are collected into a block matrix \hat{P} , the so-called *tentative prolongator*, and the upper triangular factors are collected into a block matrix that is used as a coarse representation of \mathcal{N} .

In the remainder of the paper, we will consider only linear systems arising from scalar partial differential equations. In the scalar case, \mathcal{N} has only one vector, and \hat{P} has M normalized columns, one for each aggregate. For each column k , an entry j is nonzero if and only if fine unknown j belongs to aggregate k .

The final prolongator P_i is created by applying a step of damped Jacobi to \hat{P} ,

$$P = \left(I - \frac{4\omega}{3} D^{-1} A \right) \hat{P}, \quad (3)$$

where D is the diagonal of A and ω is an estimate of the largest eigenvalue of $D^{-1}A$.

3.1 Semicoarsening

For problem with anisotropic stencils or meshes, traditional point smoothers (*e.g.*, Jacobi or Gauss-Seidel) tend to smooth only in certain directions. This can severely degrade on the convergence of the multigrid algorithm. One way to deal with this problem is a technique known as semicoarsening [23, 26], wherein coarsening is performed only in directions where the smoother is effective at damping error. For structured grid problems this may look something like “only coarsen in z at this level” or “coarsen in x and y but not z ,” depending on the nature of the anisotropy. In either case, semicoarsening allows us to recover optimal multigrid convergence while still using a traditional point smoother. However, this convergence comes at the cost of solving a larger coarse grid problem. We will consider the effects of semicoarsening on the cost of SpMM routines in more detail in Section 5.3.2.

3.2 Solve Phase

As mentioned in the introduction, algebraic multigrid methods rely on a hierarchy of increasingly coarse resolution problems of the form $A_i x_i = b_i$ called *levels* to accelerate the solution of the given linear system $Ax = b$. Applying the multigrid algorithm requires traversing the levels. Each level’s linear system is solved using a typically inexpensive solver called a *smoother* that is often based on a sparse matrix-vector kernel with the matrix A_i . Information is propagated between levels by means of the prolongators P_i and restrictors R_i . Perhaps the most common traversal scheme is called the V-cycle, in which the levels are visited from fine to coarse, and then from coarse to fine. At each level of the descent and ascent, the smoother is applied, and restrictors and prolongators propagate information between levels via sparse-matrix vector applications. Thus, the running time of the solve phase depends heavily on the efficiency of the sparse-matrix vector kernel with matrices A_i , P_i , and R_i .

We do not consider the solve phase in this work except to point out that our proposed approach for the setup phase has one small (positive) effect on the solve phase. The running time of a sparse matrix-vector kernel depends on the distribution of the nonzeros of the sparse matrix, and we propose using a different distribution of the R matrix than the standard approach. This change is considered in more detail (along with communication analysis) in Section 5.3.3.

4 Parallel 1D Algorithms for SpMM

We restrict attention to “1D” SpMM algorithms, as defined in [4]. Such algorithms align naturally with 1D matrix partitions, assigning work to processors by subdividing only one dimension of the three dimensional computation cube representing matrix multiplication. Because there are three dimensions to subdivide, there are three types of 1D algorithms: row-wise, column-wise, and outer-product algorithms. These algorithms can be applied to general sparse matrices, and we will use the notation $A \cdot B = C$ to reference the input and output matrices. See Figure 1 for a visualization of these three types of algorithms.

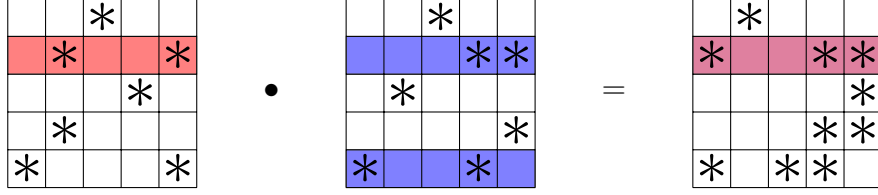
Our main motivation for restricting attention to 1D algorithms is to leverage the existing software infrastructure in the Trilinos software package, which assumes 1D distribution of matrices. However, there are other reasons to expect that 1D algorithms will be effective in this setting. In particular, exploiting the sparsity structure of the input matrices is more straightforward in the 1D case (as described below), resulting in communication patterns of halo exchanges. The most widely used 2D algorithms are Sparse Cannon and Sparse SUMMA (as discussed in Section 2), which have communication patterns that ignore the sparsity structure of the matrices. For matrices corresponding to 2D and 3D physical problems, 1D algorithms involve exchanging messages with a processor’s nearest neighbors (*e.g.*, 8 or 26 other processors on a structured grid, independent of the total number of processors p), whereas Sparse Cannon and Sparse SUMMA involve at least \sqrt{p} messages. While it is possible to exploit sparsity structure within 2D or 3D algorithms (see [4]), we are not aware of any robust implementation of such an algorithm. We note that deep in the multigrid hierarchy, coarse grids typically fill-in and lose some of the properties of the physical structure, and the advantages of structure-exploiting 1D algorithms can deteriorate.

Some evidence for the row-wise algorithm’s communication efficiency for one of the sparse matrix multiplications ($A \cdot P$) within the Galerkin triple product is presented in [5]. The authors show that the row-wise algorithm applied to a regular 3D grid with typical matrix distributions achieves a communication cost almost as low as the best 3D algorithm identified by a hypergraph partitioner.

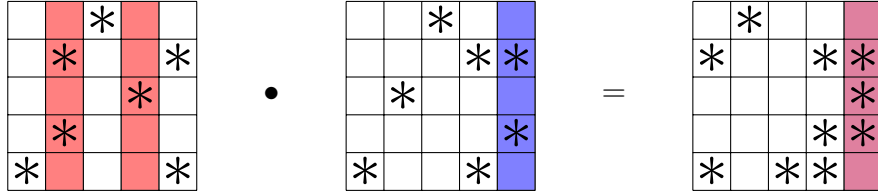
4.1 Row-Wise Algorithm

The atomic task in the row-wise algorithm is the computation of a row of the output matrix (see Figure 1(a)). Here, the assumed data distribution of all three matrices is row-wise, so that each row is owned entirely by one processor. Furthermore, we assume that A and C have identical row distributions.

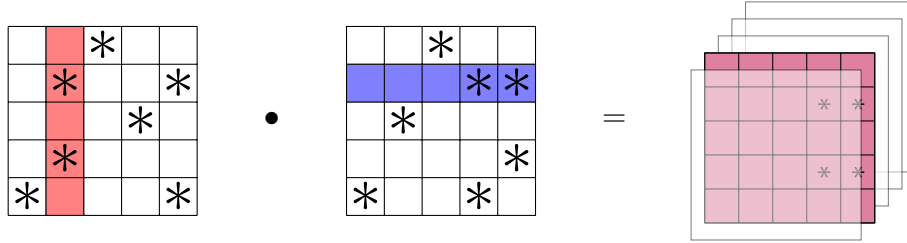
If the row-wise algorithm is used to compute the product $A \cdot B = C$, only entries of B are communicated. Each processor owns a subset of the rows of A , and in order to compute the same rows of C , each processor needs to access the rows of B , some of which are not local, corresponding to nonzero columns in its local rows of A .



(a) Row-wise algorithm: Rows of C are computed as linear combinations of rows of B . Row i of C depends on row i of A and rows of B corresponding to nonzero columns in row i of A . The parallel algorithm uniquely assigns each row of C to a processor.



(b) Column-wise algorithm: Columns of C are computed as linear combinations of columns of A . Column j of C depends on column j of B and columns of A corresponding to nonzero rows in column j of B . The parallel algorithm uniquely assigns each column of C to a processor.



(c) Outer-product algorithm: C is computed as a sum of rank-one outer products of columns of A and corresponding rows of B . The k th outer product depends on column k of A and row k of B . The parallel algorithm uniquely assigns each outer product to a processor.

Figure 1. The three 1D algorithms for SpMM. Asterisks represent nonzero values, and highlighted submatrices represent a subset of the computation in each algorithm that is uniquely assigned to a processor.

Thus, the total number of entries a processor must receive is roughly the number of nonzero columns in local rows of A corresponding to nonlocally owned rows of B times the average nnz per row of B . We define the indices corresponding to nonzero columns of local rows of A and nonlocal rows of B as the *halo* of a given processor, and the number of entries a processor must receive is the number of halo rows times the average nnz per halo row of B , which might differ from the overall average.

4.2 Column-Wise Algorithm

The column-wise algorithm has very similar characteristics to the row-wise algorithm, though the atomic task is the computation of a column of the output matrix (see Figure 1(b)). In this case, the assumed data distribution is column-wise, we assume that B and C have identical distributions, and only entries of A are communicated. Each processor owns a subset of the columns of B , and in order to compute the same columns of C , each processor needs to access the columns of A , some of which are not local, corresponding to nonzero rows in its local columns of B . In this case, we define the halo of a processor as the indices corresponding to nonzero rows of local columns of B and nonlocal columns of A , and the number of entries a processor must receive is the number of halo columns times the average nnz per halo column of A , which might differ from the overall average.

4.3 Outer-Product Algorithm

In the case of the outer-product algorithm, the atomic task is the outer product of a column of A and the corresponding row of B , as shown in Figure 1(c). For this algorithm, we assume that A is distributed column-wise, B is distributed row-wise, and that those distributions match (so that for each i , column i of A and row i of B are owned by the same processor). The desired distribution of C can be arbitrary, but here we will assume it is row-wise.

If the outer-product is used to compute the product $A \cdot B = C$, only (possibly unreduced) entries of C are communicated. Each processor owns a subset of columns of A and corresponding rows of B , but cannot completely compute all local entries of C ; the processor needs to receive contributions to local entries from other processors and merge them with local contributions. As with the row-wise algorithm, we can define the set of halo indices for a given processor. In the case of the outer-product algorithm, the halo indices are rows of A corresponding to local rows of C that include a nonzero in a nonlocal column.

Matrix	nnz	rows	cols	nnz/row	nnz/col
A	$N \cdot 3^d$	N	N	3^d	3^d
\hat{P}	N	N	$N/3^d$	1	3^d
P	$N \cdot (5/3)^d$	N	$N/3^d$	$(5/3)^d$	5^d
R	$N \cdot (5/3)^d$	$N/3^d$	N	5^d	$(5/3)^d$
AP	$N \cdot (7/3)^d$	N	$N/3^d$	$(7/3)^d$	7^d
A_c	N	$N/3^d$	$N/3^d$	3^d	3^d

Table 1. Matrix statistics for 3^d -point stencil fine grid operator and ideal coarsening. Reported nonzero counts ignore boundary effects, so actual counts approach the reported values as $n \rightarrow \infty$. Similarly, nnz/row and nnz/col columns are averages (in the limit).

5 Communication Cost Analysis for 3^d -point Stencil

In order to analyze the communication costs of SpMM routines in the triple product coarsening operation within algebraic multigrid, we first introduce a graph-theoretic interpretation of the relevant matrices. For the fine grid problem, we consider a regular mesh with n points in each dimension, which means that our total number of fine grids points is n^d , where d is the number of dimensions.

5.1 Matrix Statistics

Since our parallel sparse matrix distribution is a 1D row distribution (*i.e.*, each row is owned entirely by one processor), and communication within our algorithms occurs in units of entire rows, we are interested in the average number of nonzeros (nnz) per row of these matrices. To compare with other column-based algorithms, we are also interested in the average nnz per column. For the case of 3^d -point stencils and ideal coarsening, we can express the limits of these quantities in closed form. Lowest order tensor-product nodal finite elements (*i.e.*, quads and hexes) have a 3^d point stencil on a regular mesh, which is why we analyze this case. We summarize the statistics of the matrices in Table 1, and derive them in more detail in the following sections.

5.1.1 Fine Grid Operator A

Because A acts on the fine grid, its dimensions are $N \times N = n^d \times n^d$. Entry A_{ij} is nonzero if the fine grid point i is adjacent to (a successor of) the fine grid point j in the graph corresponding to A ; in other words, A_{ij} is nonzero if, when A acts on the fine grid, the

output value at point i depends on the input value at point j .

In the case of a 3^d -point stencil, every non-boundary node is adjacent to 3^d neighbors and all connections are reciprocal. Thus, the average nnz per row of A approaches 3^d as $n \rightarrow \infty$.

Note that we do not assume that A is a symmetric matrix, though it is structurally symmetric here in the case of a 3^d -point stencil. This implies that the average nnz per col also approaches 3^d .

5.1.2 Tentative Prolongation Operator \hat{P}

A prolongation (or interpolation) operator maps the coarse grid to the fine grid, so the dimensions of \hat{P} are $N \times (N/3^d)$. The *tentative* prolongation operator maps each coarse grid aggregate to a disjoint set of fine grid nodes (its “members”). Entry \hat{P}_{ij} is nonzero if the fine grid node i is a member of coarse grid aggregate j . As each fine grid node is a member of exactly one aggregate, the average nnz per row of \hat{P} is exactly 1.

Here we assume ideal coarsening. That is, we assume the fine grid to be a regular $(3\ell)^d$ mesh, for some positive integer ℓ , so that we can pick as aggregate roots fine grid nodes all of whose coordinates are congruent to 1 mod 3. The members of an aggregate include the root and all of the root’s neighbors (in the graph corresponding to A). Since ℓ is a positive integer, all fine grid nodes are adjacent to some root and are therefore a member of an aggregate. In the case of a 3^d -point stencil with ideal coarsening, every aggregate has exactly 3^d members, so the nnz in each column of \hat{P} is exactly 3^d .

5.1.3 Prolongation Operator P

Similar to \hat{P} , the prolongation operator P has dimensions $N \times (N/3^d)$. We assume that P is computed from \hat{P} via Jacobi smoothing: $P = (I - \omega D^{-1}A)\hat{P}$, where ω is a scalar and $D = \text{diag}(\text{diag}(A))$. Thus, the sparsity structure of P is equivalent to that of $A\hat{P}$. Entry P_{ij} is nonzero if the fine grid node i is adjacent (in the graph corresponding to A) to a fine grid node that is a member of coarse grid aggregate j .

In the case of a 3^d -point stencil, the nnz of a given row of P depends on the position of the fine grid node within the aggregate. For example, a root node is adjacent only to nodes which are members of its own aggregate, so the nnz in a row corresponding to a root node is 1. However, a fine grid node in the corner of an aggregate may be adjacent to other nodes from multiple aggregates. For $d = 1$, there are two types of nodes: roots and non-roots. We can represent the possibilities for nnz per row as

$$\begin{bmatrix} 2 & 1 & 2 \end{bmatrix}$$

where we have labeled each fine grid node within one aggregate (not on the physical boundary). That is, in the $d = 1$ case, non-root nodes at the boundary of an aggregate are adjacent

to member nodes of two different aggregates, while a root node is adjacent to nodes that are all members of the same aggregate. Since all non-boundary aggregates show this pattern, the average nnz per row approaches $5/3$ as $n \rightarrow \infty$ for $d = 1$. For $d = 2$, the pattern is

$$\begin{bmatrix} 4 & 2 & 4 \\ 2 & 1 & 2 \\ 4 & 2 & 4 \end{bmatrix}$$

and for $d = 3$ the pattern is

$$\begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 8 & 4 & 8 \end{bmatrix} \quad \begin{bmatrix} 4 & 2 & 4 \\ 2 & 1 & 2 \\ 4 & 2 & 4 \end{bmatrix} \quad \begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 8 & 4 & 8 \end{bmatrix}$$

where we have flattened the $3 \times 3 \times 3$ aggregate into 3 matrices. Computing averages over a given aggregate, we see that the average nnz per row approaches $(5/3)^d$ as $n \rightarrow \infty$ for $d \in \{1, 2, 3\}$. Note that the average nnz per col of P is 5^d .

5.1.4 Restriction Operator R

We assume here that the restriction operator R has the same sparsity structure as the transpose of the prolongation operator, P^T . Thus the average nnz per row of R approaches 5^d and the average nnz per col of R approaches $(5/3)^d$ as $n \rightarrow \infty$.

5.1.5 Intermediate Matrix AP

One way to compute the triple product $A_c = RAP$ is to perform the rightmost multiplication first, yielding an intermediate matrix AP . If we assume that R has the same sparsity structure as P^T , then similar analysis will apply to computing RA first. The dimensions of AP are $N \times (N/3^d)$. Note that since the sparsity structure of P is equivalent to $A\hat{P}$, the sparsity structure of AP is equivalent to $A^2\hat{P}$. Thus, entry $(AP)_{ij}$ is nonzero if the fine grid node i is 2-hop adjacent to (*i.e.*, is a successor of a successor of) a member of coarse grid aggregate j .

In the case of a 3^d -point stencil, the nnz of a given row of AP again depends on the position of the fine grid node within the aggregate. For example, given two hops, a root node can now reach all 3^d aggregates adjacent to its own aggregate. Using the same notation as before, the possible nnz in a row (for fine grid nodes in non-boundary aggregates) are as follows. The patterns are for $d = 1$:

$$\begin{bmatrix} 2 & 3 & 2 \end{bmatrix}$$

for $d = 2$:

$$\begin{bmatrix} 4 & 6 & 4 \\ 6 & 9 & 6 \\ 4 & 6 & 4 \end{bmatrix}$$

and for $d = 3$:

$$\begin{bmatrix} 8 & 12 & 8 \\ 12 & 18 & 12 \\ 8 & 12 & 8 \end{bmatrix} \quad \begin{bmatrix} 12 & 18 & 12 \\ 18 & 27 & 18 \\ 12 & 18 & 12 \end{bmatrix} \quad \begin{bmatrix} 8 & 12 & 8 \\ 12 & 18 & 12 \\ 8 & 12 & 8 \end{bmatrix}$$

so the average nnz per row is given in general by $(7/3)^d$. Note that the average nnz per col of AP is 7^d .

5.1.6 Coarse Grid Operator A_c

Because the coarse grid operator A_c acts on the coarse grid, its dimensions are $(N/3^d) \times (N/3^d)$. The matrix coarse grid operator is determined by $A_c = RAP$. Intuitively, entry $(A_c)_{ij}$ is nonzero if the coarse grid aggregate i is adjacent to (*i.e.*, a successor of) the coarse grid aggregate j in the graph corresponding to A_c . Assuming A is structurally symmetric and $R = P^T$, we can characterize an entry of A_c more specifically: $(A_c)_{ij}$ is nonzero if coarse grid aggregate i has a member that is 3-hop adjacent to (*i.e.*, a successor of a successor of a successor of) a member of coarse grid aggregate j .

In the case of a 3^d -point stencil, 3 hops is not enough to cross an entire aggregate. Thus, an aggregate is adjacent to only those aggregates that share 1-hop member neighbors. This implies that the structure of A_c is equivalent to that of A : A_c is also a 3^d -point stencil matrix, and thus the average nnz per row and average nnz per col also approach 3^d . We note that because of the reduction in matrix dimension, the true averages will not be as close to 3^d as those of A ; boundary effects play a larger role for smaller problems.

5.2 Communication Cost Analysis for AMG Setup

We will consider various algorithms for computing A_c for a given fine grid A and tentative prolongation operator \hat{P} . The usual approach is to compute

1. $P = (I - \omega D^{-1}A) \cdot \hat{P}$,
2. $AP = A \cdot P$, and
3. $A_c = R \cdot (AP)$.

	Row-Wise	Column-Wise	Outer-Product
$P = A \cdot \widehat{P}$	1	0	$(5/3)^{d-1}$
$AP = A \cdot P$	$2(5/3)^{d-1}$	3^d	$2(7/3)^{d-1}$
$A_c = R \cdot (AP)$	$2(7/3)^{d-1}$	$2(5/3)^d$	2

Table 2. Communication costs of 1D algorithms for one level of algebraic multigrid setup for 3^d -point stencil matrix. Each entry is the maximum per-processor bandwidth cost (number of elements sent and received), divided by $2h_F$ (where h_F is the size of the outer halo) and ignoring lower order terms, for the given computation and algorithm. Each entry assumes the matrices are distributed as required by the corresponding algorithm (the outer product analysis also assumes the output matrix is distributed row-wise). The cost of the proposed choice of algorithm for each row is highlighted in boldface.

For symmetric problems (or slightly nonsymmetric problems) the R matrix is usually generated by $R = P^T$. We assume A is numerically symmetric and $R = P^T$ in this section. For problems with pronounced asymmetry, $R = \widehat{P}^T (I - \omega D^{-1}A)$ might be chosen instead. See Section 5.3.1 for a discussion of this case.

In the case of 3^d -point stencil with ideal coarsening, we assume the number of processors p can be arranged into a d -dimensional logical grid, with $p^{1/d}$ dividing n evenly. Then we assume the partitioning of the fine and coarse grids to processors follows the regular geometric pattern so that, in the 3D case, each processor owns a contiguous $(n/p^{1/3}) \times (n/p^{1/3}) \times (n/p^{1/3})$ subset of the fine grid. Furthermore, we assume no aggregate crosses a processor boundary (*i.e.*, we assume uncoupled aggregation).

Table 2 summarizes the results of this section. We state the communication costs for the three SpMM computations within AMG setup for the three different 1D algorithms. We emphasize in boldface the proposed algorithm for each SpMM. Note that although the column-wise approach requires no communication for $A \cdot \widehat{P}$, we do not use it for the SpMM because it would require a re-distribution of P from column-wise to row-wise in order to perform the row-wise algorithm for $A \cdot P$. This redistribution requires more communication than performing the row-wise algorithm on $A \cdot \widehat{P}$. The row-wise algorithm is clearly the optimal choice for the second SpMM, and the outer-product algorithm is the optimal choice for the third SpMM. The analysis for the row-wise algorithm on all SpMMs is given in Section 5.2.1, the analysis for the outer-product algorithm on the final SpMM is given in Section 5.2.2, and a summary of the analysis for the other entries are given in Appendix A.

5.2.1 Row-Wise Approach to Triple Product

Here we consider the communication costs of using the row-wise algorithm for each SpMM. The costs of each of the SpMM operations will depend on the size of the halo with respect to the fine grid operator. In the 3D case, if a processor owns a contiguous $(n/p^{1/3}) \times (n/p^{1/3}) \times (n/p^{1/3})$ subset of the fine grid, the halo will be all grid nodes that are adjacent to (but not in) the subset. We will refer to this as the outer halo, as the nodes are not local; the inner halo corresponds to data that must be sent to other processors. Note that we ignore processors on the processor grid boundary, as their halos are smaller. The number of outer halo nodes h_F is then

$$h_F = \left(\left(\frac{N}{p} \right)^{1/3} + 2 \right)^3 - \frac{N}{p} = 6 \left(\frac{N}{p} \right)^{2/3} + 12 \left(\frac{N}{p} \right)^{1/3} + 8,$$

where the first term corresponds to faces, the second to edges, and the third to corners. In the general case, we have

$$h_F = 2d \left(\frac{N}{p} \right)^{1-1/d} + O \left(\left(\frac{N}{p} \right)^{1-2/d} \right),$$

and the number of inner halo nodes differs by only a lower order term. In particular, when we ignore lower order terms, we are considering only halo nodes lying on boundary “faces.” In the 2D case, we are ignoring corner nodes; in the 3D case, we are ignoring edge nodes and corner nodes.

The communication costs of $P = (I - \omega D^{-1}A) \cdot \hat{P}$ are identical to $A \cdot \hat{P}$, so we first consider $A \cdot \hat{P}$. For the row-wise algorithm, the number of nonlocal rows of \hat{P} needed is h_F , and the nnz of each row of \hat{P} is 1. Thus, the total number of nonlocal elements received by a non-boundary processor is h_F , and that processor must send as many elements to its neighbors.

The second multiply is $AP = A \cdot P$. Again, the left input matrix is A , so the number of rows needed here is also h_F . The average nnz per row of P is $(5/3)^d$, but that average is not an accurate measure if we restrict attention to halo rows. For example, in the 1D case, the pattern of nnz in a row corresponding to fine grid nodes within an aggregate is $\begin{bmatrix} 2 & 1 & 2 \end{bmatrix}$. Because aggregates do not traverse processor boundaries, a root node can never be in a halo. Thus, in the 1D case, the nnz of every halo row of P is 2. In the 2D case, the nnz of halo node rows follows the pattern $\begin{bmatrix} 4 & 2 & 4 \end{bmatrix}$. In the 3D case, the pattern is

$$\begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 8 & 4 & 8 \end{bmatrix}.$$

In general, the average nnz per row (restricted to the halo) of P is $2(5/3)^{d-1}$, and the total number of nonlocal elements received is $h_F \cdot 2(5/3)^{d-1}$. Again, the number of elements sent is the same as the number received.

In order to perform the row-wise algorithm to compute $R \cdot (AP)$, we require the R matrix be distributed in the same row distribution as A_c . A local transposition of each processor's local rows of P yields R in column distribution, but to attain row distribution, a communication operation similar to a SpMM is required. While a row distribution of P means a processor owns all weights of edges incident to local fine nodes, a row distribution of R means a processor owns all weights of edges incident to local coarse aggregates. Thus, the nonlocal data needed lies in rows of P corresponding to a processor's halo. Since all the halo rows of P were already accessed to perform $AP = A \cdot P$, all of the data needed to attain row distribution of R is available locally, and no more communication is required. Note that this optimization is not possible when coupled aggregation is used (in which case aggregates can cross processor boundaries).

Note that if R is explicitly redistributed from column-wise to row-wise, then each processor needs to receive nonzeros corresponding to edges from local fine grid points to nonlocal coarse grid aggregates. The average number of nonlocal coarse grid aggregates to which a local fine grid point is adjacent is $(5/3)^{d-1}$. In the 2D case, the pattern is $\begin{bmatrix} 2 & 1 & 2 \end{bmatrix}$, and in the 3D case, the pattern is

$$\begin{bmatrix} 4 & 2 & 4 \\ 2 & 1 & 2 \\ 4 & 2 & 4 \end{bmatrix}.$$

Thus, the communication cost of the explicit redistribution is $h_F \cdot (5/3)^{d-1}$, or half the cost of the $AP = A \cdot P$ SpMM.

The last multiply is $A_c = R \cdot (AP)$. The halo required for this operation is with respect to R , but because of how $R = P^T$ is computed (so that the row distribution of R matches that of A), this halo is the same as that of A and consists of h_F rows. Again, the average nnz in halo rows of AP differs from the overall average. The patterns for nnz in halo rows of AP are given by 2 (for 1D), $\begin{bmatrix} 4 & 6 & 4 \end{bmatrix}$ (for 2D), and

$$\begin{bmatrix} 8 & 12 & 8 \\ 12 & 18 & 12 \\ 8 & 12 & 8 \end{bmatrix}$$

(for 3D). For AP , the average nnz is $2(7/3)^{d-1}$ and therefore the total number of nonlocal elements received is $h_F \cdot 2(7/3)^{d-1}$ (and the same number of elements must be sent by each processor).

5.2.2 Outer-Product Approach to $A_c = R \cdot (AP)$

An alternative method to the one given in Section 5.2.1 is to use the outer-product algorithm for the final multiplication. The first observation is that since $R = P^T$, a transpose of local P data achieves R in a global column distribution that matches the row distribution of AP . Thus, the assumptions on data distribution for the outer-product algorithm are met without any communication. The next observation is that the total nnz of A_c is less than the total

nnz of AP , so communicating A_c may be cheaper than communicating AP (*i.e.*, the outer-product algorithm may be cheaper than the row-wise algorithm). More precisely, we will see that although the average nnz per row of A_c is greater than that of AP , fewer (unreduced) rows of A need to be communicated because the halo in the coarse grid is smaller than the halo in the fine grid.

In order to determine the communication cost of the outer-product algorithm, we will first compute the size of the halo. The desired distribution of A_c is row-wise, with each processor owning rows corresponding to coarse grid aggregates comprised of local fine grid nodes. Thus, the halo consists of rows of R corresponding to local aggregates that are adjacent to nonlocal fine grid nodes (since the local columns of R correspond to local fine grid nodes). That is, the halo is the set of local aggregates at the processor boundary, an inner halo in the coarse grid. The outer halo corresponds to data that must be sent from the processor to its neighbors. The number of inner halo aggregates h_C is then

$$h_C = \frac{N/3^d}{p} - \left(\left(\frac{N/3^d}{p} \right)^{1/d} - 2 \right)^d = 2d \left(\frac{N/3^d}{p} \right)^{1-1/d} + O \left(\left(\frac{N}{p} \right)^{1-2/d} \right) \approx \frac{h_F}{3^{d-1}}.$$

Again, the size of the outer halo differs by a lower order term.

While the average nnz per row of A_c is 3^d , the average is smaller for unreduced rows of A_c corresponding to halo aggregates. The nnz per row of A_c corresponds to the number of aggregates reachable from a given aggregate with three hops in the fine grid. The nnz per unreduced row of A_c corresponds to the number of aggregates reachable from a given inner halo aggregate with three hops, provided that the first hop is to a fine grid node in the outer halo. Because of the constraint on the first hop, the nnz per unreduced row of A_c is less than 3^d (the average nnz per row of the final A_c). Consider a halo aggregate on a boundary face. In the 1D case, a right-boundary aggregate can reach the aggregate to its right (across the boundary) and itself, but not its neighbor to the left, so the unreduced row of A_c has 2 nonzeros. In the 2D (3D) case, an aggregate can reach its 3 (9) neighbor aggregates across the boundary, and those neighbors also on the boundary, but not its 3 (9) neighbor aggregates in the opposite direction from the boundary, so its row has 6 (18) nonzeros. See Figure 2 for an illustration in the 2D case of the 6 aggregates reachable from a halo aggregate given the constraint on the first hop. Thus, the nnz per unreduced row is $2 \cdot 3^{d-1}$. The total number of nonlocal elements received is then $h_C \cdot 2 \cdot 3^{d-1} \approx h_F \cdot 2$, and the same number of elements must be sent to neighbors.

Note that this cost is a factor of $(7/3)^{d-1} \approx 5.44$ smaller than the cost of the row-wise algorithm.

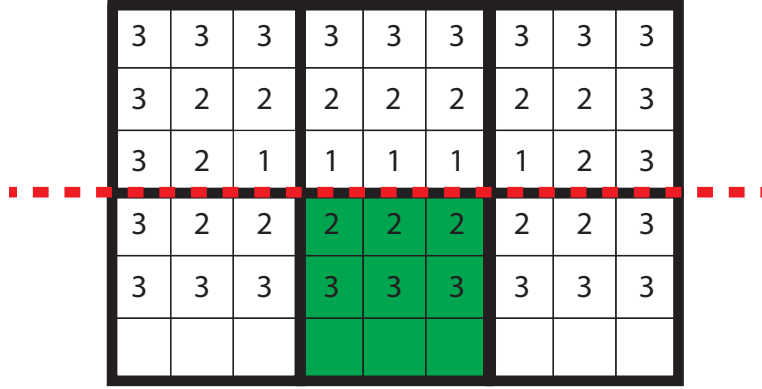


Figure 2. Illustration of six aggregates (boxes with thick borders) reachable from the shaded aggregate on a processor boundary (shown as a dashed line), given three hops on the 2D 9-pt stencil fine grid (one for the smoothing of P , one for the A in the $A_c = RAP$ product and one for the smoothing of R). Here, the first hop is always constrained to cross the processor boundary and the fine grid unknowns are assigned a number based on the smallest number of hops needed to reach the unknown from the shaded aggregate. The number of reachable aggregates corresponds to the sparsity pattern of the (shaded) row of A_c . These (unreduced) nonzeros must be communicated in the outer-product algorithm. We see that in this idealized case, the communicated nonzeros correspond to only 6 of 9 neighboring aggregates.

5.3 Extensions of Theoretical Analysis

5.3.1 Nonsymmetric Problems

In the case that A is nonsymmetric, there may be another SpMM to compute $R = \hat{P}^T (I - \omega D^{-1} A)$. As in the symmetric case, we consider the three 1D algorithms, this time for computing $R = \hat{P}^T \cdot A$.

For 3^d -point stencil analysis, we will consider A to be structurally symmetric but not numerically symmetric. Since this computation is the structural transpose of $P = A \cdot \hat{P}$, the communication costs are given in the first row of Table 2. That is, assuming the input matrices are appropriately distributed, the row-wise algorithm has a communication cost of approximately $2 \cdot 3^d \cdot h_F$, the column-wise algorithm has a cost of $2 \cdot h_F$, and the outer-product algorithm has a cost of $2 \cdot (5/3)^{d-1} \cdot h_F$ (assuming the output is distributed column-wise).

Although this analysis suggests that the column-wise algorithm should be used to compute R , the communication cost assumes that A is distributed column-wise. The optimal algorithm for computing $P = A \cdot \hat{P}$ is the row-wise algorithm, which requires A to be distributed row-wise. Thus, in order to use the best algorithm for each computation, an explicit redistribution of A is required. Note that the sparsity structure of \hat{P} is such that row-wise and column-wise distributions are the same (this assumes uncoupled aggregation). The communication cost of redistributing A from row-wise to column-wise is $2 \cdot 3^{d-1} \cdot h_F$, since every halo point is adjacent to 3^{d-1} neighbors across the processor boundary.

Note that the outer-product algorithm for $R = \hat{P}^T \cdot A$ assumes \hat{P}^T is distributed column-wise and A is distributed row-wise, which matches the requirements of the row-wise algorithm for computing $P = A \cdot \hat{P}$. Furthermore, the outer-product algorithm outputs R in a column-wise distribution, which matches the requirements of the outer-product algorithm for $A_c = R \cdot (AP)$. Because the cost of explicitly redistributing A exceeds the cost of the outer-product algorithm, the outer-product algorithm is the optimal choice of 1D algorithms for computing R in the case of the 3^d -point stencil matrix.

5.3.2 Semi-Coarsening

In the case of semi-coarsening (see Section 3.1), the ratio of fine grid nodes to coarse grid aggregates drops from 9 to 3 (in the 2D case) or from 27 to 9 or 3 (in the 3D case, depending on how many dimensions are coarsened). Assuming the same distribution of fine grid nodes to processors as in the previous section, we can change the dimensions and sparsity structure of \hat{P} and repeat the analysis to determine the most efficient algorithms for the coarse grid setup. To prevent extra fill in the coarse grid operator, we also use filtering in computing the prolongation operator P . That is, we compute $P = (I - (4/3)\omega D^{-1} A_F) \hat{P}$, where A_F is a filtered representation of A that includes only those nonzeros corresponding to edges in the coarsened directions. The communication requirements of computing P in this way follows that of computing $A_F \cdot \hat{P}$. In this case, the analysis differs from Section 5.2 and

full coarsening. We provide a comparison of the communication costs for the final $R \cdot (AP)$ multiply for row-wise and outer-product algorithms in this section. We consider the 2D case with coarsening in one dimension and the 3D case with coarsening in one or two dimensions.

2D Case - One Coarsened Dimension We first consider the 2D 9-point stencil case, where coarsening occurs in only one dimension. In this case, if A is $N \times N$, then \hat{P} is $N \times (N/3)$. While \hat{P} has 3 nonzeros per column, P has the structure of $A_F \hat{P}$ and has 5 nonzeros per column. The 5 nonzeros in a column correspond to an aggregate's 3 member fine points and the 2 neighbor fine points in the coarsened direction. The matrix AP has 21 nonzeros per column, and the nnz per row follows the pattern $\begin{bmatrix} 6 & 9 & 6 \end{bmatrix}$.

If we compute $A_c = R \cdot (AP)$ using the row-wise algorithm, then the communication includes receiving nonlocal rows of AP that are needed to compute rows of A_c corresponding to local aggregates. Because $R = P^T$ has the structure of $\hat{P}^T A$, the only required nonlocal rows correspond to halo points owned by neighboring processors in the coarsened dimension. That is, only 2 out of the 4 processor boundary edges involve communication. Furthermore, all rows of AP corresponding to points in this part of the halo have 6 nonzeros per row. Thus, the communication cost of the row-wise algorithm is $(h_F/2) \cdot 6 = 3 \cdot h_F$.

If we use the outer-product algorithm to compute $A_c = R \cdot (AP)$, then the communication involves receiving unreduced local rows of A_c computed by other processors. These nonlocal unreduced rows correspond to paths that start from a local aggregate, hop to a nonlocal point in the graph corresponding to R , hop to another point in the graph corresponding to A , and finally hop to another aggregate in the graph corresponding to P . The number of local aggregates for which such a path is possible is $h_F/2$, because a hop across a processor boundary in the graph corresponding to R must be in the coarsened direction and there are as many coarse grid aggregates along those two boundary edges as there are fine points. The number of nonzeros in each of the unreduced rows is the number of aggregates reachable by an inner halo aggregate under the constraints of this 3-hop path, which is 6. We illustrate these 6 reachable aggregates in Figure 3. Therefore, the communication cost of the outer-product algorithm is also $3 \cdot h_F$, the same as the row-wise algorithm.

3D Case - Two Coarsened Dimensions We now consider the 3D 27-point stencil case, where only two dimensions are coarsened. Here \hat{P} is $N \times (N/9)$ and each column has 9 nonzeros. As before, P has the structure of $A_F \hat{P}$ and so has 25 nonzeros per column; AP has 147 nonzeros per column, and the nnz per row follows the pattern

$$\begin{bmatrix} 12 & 18 & 12 \\ 18 & 27 & 18 \\ 12 & 18 & 12 \end{bmatrix}.$$

If we use the row-wise algorithm to multiply $R \cdot (AP)$, then processors communicate rows of AP corresponding to halo points in the graph corresponding to R . Because R was

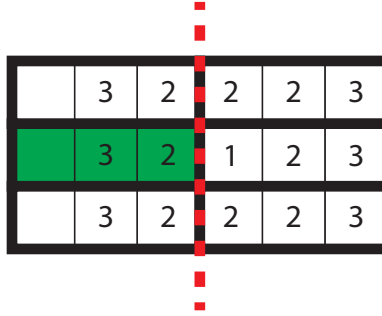


Figure 3. Illustration of six aggregates (boxes with thick borders) reachable from the 2D semicoarsened shaded aggregate on a processor boundary (shown as a dashed line), given three hops on the 2D semicoarsening 9-pt stencil fine grid (one for the smoothing of P with vertical connections dropped, one for the A in the $A_c = RAP$ product with all connections included, and one for the smoothing of R with vertical connections dropped). Here, the first hop is always constrained to cross the processor boundary and the fine grid unknowns are assigned a number based on the smallest number of hops needed to reach the unknown from the shaded aggregate. The number of reachable aggregates corresponds to the sparsity pattern of the (shaded) row of A_c . These (unreduced) nonzeros must be communicated in the outer-product algorithm. We see that in this idealized case, the communicated nonzeros correspond to only 6 of 9 neighboring aggregates.

computed with the filtered matrix A_F , these halo points lie along faces in the two coarsened dimensions, but not in the third dimension. Thus, the number of rows each processor needs to receive is approximately $2h_F/3$. The average number of nonzeros in these rows is $(12 + 18 + 12)/3 = 14$, yielding a total communication cost of $(28/3) \cdot h_F$.

Using the outer-product algorithm for $A_c = R \cdot (AP)$, each processor must receive unreduced rows of A_c from other processors. As in the 2D case, such rows correspond to 3-hop paths that start from a local aggregate and meet the constraints of the graphs corresponding to R , A , and P . The number of local aggregates (or unreduced rows) is the size of the inner aggregate halo that lies along faces in the coarsened directions. That is, only 4 out of 6 faces are included in the halo, and on these faces the number of aggregates is $1/3$ of the number of fine points. Thus, the number of unreduced rows that must be received approaches $2h_F/9$. The number of nonzeros in each of the unreduced rows is the number of aggregates reachable by an inner halo aggregate under the constraints of this 3-hop path, which is 18. The communication cost of the outer-product algorithm is thus $4 \cdot h_F$, which is a factor of $7/3$ less than the row-wise algorithm.

3D Case - One Coarsened Dimension Finally, we consider the 3D 27-point stencil case, where only one dimension is coarsened. The analysis follows the previous cases: AP is $N \times (N/3)$ and the nnz per row follows the pattern $[18 \ 27 \ 18]$. In the row-wise algorithm, each processor must receive approximately $h_F/3$ rows from other processors. Each of those rows has 18 nonzeros, so the communication cost approaches $6 \cdot h_F$. In the outer-product algorithm, each processor must receive approximately the same number of (unreduced) rows, and each of those rows contains the same number of nonzeros. Thus, the communication costs of the two algorithms are equivalent in this case, ignoring lower order terms.

5.3.3 Applying R during Solve Phase

An important benefit of using the outer-product algorithm for $A_c = R \cdot (AP)$ is that the R matrix need not be explicitly redistributed into row-wise distribution. However, as described in Section 3.2, the solve phase includes computing SpMV products with R , A , and P . If the row-wise approach is used for all SpMM operations within the setup phase, then all three matrices are available in row-wise distribution and the same row-wise SpMV algorithm can be used for all during the solve phase. If the R matrix is distributed column-wise (as is the case in the outer-product approach), then a column-wise SpMV algorithm must be used for R during the solve phase. The following analysis argues that this is yet another benefit of the outer-product approach: in the case of applying R , column-wise SpMV is more communication efficient than row-wise SpMV.

Let us assume R is computed from a 3^d -point stencil fine grid operator as described in Section 5.1.4. If R is row-wise distributed, then the communication cost of SpMV can be computed similarly to the row-wise SpMM algorithm for $R \cdot (AP)$, assuming the output vector is distributed according to the row distribution of R . That is, the number of rows

of the input vector that must be read by a processor is the size of that processor's fine grid halo (with respect to R), which is h_F . The average nnz in halo rows of the input vector is exactly one (because it is a dense vector), so the communication cost of the row-wise SpMV is h_F .

On the other hand, if R is column-wise distributed, then the communication cost of SpMV can be computed similarly to the outer-product SpMM algorithm for $R \cdot (AP)$, assuming the input vector is distributed according to the column distribution of R . In this case, the output vector is communicated according to each processor's coarse grid halo. As described in Section 5.2.2, the size of the coarse grid halo is $h_F/3^{d-1}$, and the nnz per row of the output vector is again one (because it is a dense vector). Thus, the communication cost of the column-wise SpMV with R is a factor of 3^{d-1} smaller than the row-wise SpMV.

5.4 All-at-once Triple Product

In this section we consider a more drastic alternative to computing the Galerkin triple product: performing one communication phase up front so that each processor has complete information locally to compute its rows of P (columns of P^T) and A_c all at once. This approach involves redundant computation but has the benefit of reducing the number of halo exchanges from three (in the case of computing P , AP , and A_c is separate calls to an SpMM routine) to one. While this decrease in the latency cost is substantial, we argue here that it comes at the expense of greater per-processor bandwidth cost (more words sent and received). We will ignore the extra computational cost.

For this analysis, we assume that A corresponds to a 3^d -point stencil (as described in Section 5.1.1) and is distributed row-wise; we seek a row-wise distribution of P that matches A and a row-wise distribution of A_c that matches that of Section 5.2. Assuming uncoupled aggregation, the local rows of \hat{P} can be computed without any communication. In order to compute the local rows of P , each processor needs access to the nonlocal rows of \hat{P} that correspond to its halo (this is identical to the SpMM of A and \hat{P}). Since this data is a subset of the data needed to compute A_c all at once, we will ignore its cost.

In order to compute the local rows of A_c , each processor needs to determine the nonzero values corresponding to edges from all coarse grid aggregates to local coarse grid aggregates. These edges are computed from all paths that originate from a local aggregate, hop to the fine grid along an edge from \hat{P}^T , take three hops in the fine grid, and then hop to a coarse grid aggregate along an edge from \hat{P} . Since the processor owns all edges corresponding to the first hop (these correspond to a locally owned row of A), it needs access to all edges corresponding to its two-hop halo with respect to the fine grid operator A . In order to account for all possibilities of the first hop, the processor also needs access to the rows of \hat{P} corresponding to the three-hop halo with respect to the fine grid operator. Ignoring lower order terms, the two-hop halo has size $2h_F$ and the three-hop halo has size $3h_F$. The processor needs entire rows of A and \hat{P} , consisting of 3^d and 1 nonzeros, respectively. Thus, the total amount of data each (non-boundary) processor needs to receive is $h_F \cdot (2 \cdot 3^d + 3)$

words, and it must send as many to other processors.

Compared to the proposed approach of computing the triple product one SpMM at a time, this all-at-once approach requires greater communication cost, by a factor of $(2 \cdot 3^d + 3)/(2 \cdot (5/3)^{d-1} + 3)$, or $3.3\times$ in the 2D case and $6.7\times$ in the 3D case. The original row-wise approach is also cheaper than the all-at-once approach, though by a smaller factor. However, if performance is latency bound, then the all-at-once approach can be beneficial as long as the increase in local computation and bandwidth cost does not outweigh the reduction in messages.

6 Performance Results

6.1 Experimental Setup

We use two experimental platforms in this study. The first is “Edison,” a Cray XC30 supercomputer located at NERSC, consisting of 5,576 dual-socket 12-core Intel “Ivy Bridge” (2.4 GHz) compute nodes. Each core has private 64KB L1 and 256KB L2 caches, and each socket has a 30MB L3 cache and 32GB of memory. The nodes are connected by a Cray “Aries” interconnect with a dragonfly topology.

The second is “Sky Bridge,” a Cray cluster at Sandia, consisting of 1,848 dual-socket 8-core Intel “Sandy Bridge” (2.6 GHz) compute nodes. Each core has private 32KB L1 and 256KB L2 caches, and each socket has a 20MB L3 cache and 32GB of memory. The nodes are connected by an Intel “QLogic QDR InfiniBand” interconnect with a fat-tree topology.

For our experiments, we use the MPI-based implementations of these algorithms in the Trilinos library [18]. Specifically, we use the algebraic multigrid package MueLu [22] and the sparse linear algebra package Tpetra. Both the row-wise and outer-product SpMM algorithms are implemented in Tpetra and used by MueLu as described in the sections below. We focus our attention on the highest level of the multigrid hierarchy (where A is the original operator), as that dominates the run time of the entire setup for these problems.

6.2 Model Problems

In this section we consider problems with structured grids and operators corresponding to regular stencils. We confirm the theoretical analysis from Section 5.2 with respect to communication costs and also test the effect of reduced communication on actual runtime.

As in Section 5.2, we assume a fine grid consisting of n^d points, where $d \in \{2, 3\}$ is the dimension of the problem. We consider four stencils: 2D 5-point, 2D 9-point, 3D 7-point, and 3D 27-point. Note that the theoretical analysis applies to the 2D 9-point and 3D 27-point stencils. For the 2D 9-point problem, we also consider semi-coarsening in one dimension (2D 9-point (semi)); for the 3D 27-point problem, we consider semi-coarsening where we do not coarsen in one (3D 27-point (semi1)) or two (3D 27-point (semi2)) dimensions. We perform weak-scaling experiments, maintaining approximately 100,000 fine grid points per core for all problems. For the 2D problems, we use numbers of nodes that are perfect squares up to a maximum of $25^2 \cdot 24 = 15000$ cores. For the 3D problems, we use numbers of nodes that are perfect cubes up to a maximum of $9^3 \cdot 24 = 17496$ cores.

We present the relative performance of the outer-product algorithm compared to the row-wise algorithm for $R \cdot (AP)$ for all model problems in Figure 4. The time for the row-wise algorithm includes the time spent redistributing R from column-wise to row-wise distribution. The largest speedup of $2.3\times$ is attained for the 3D 27-point problem running on 17,496 cores.

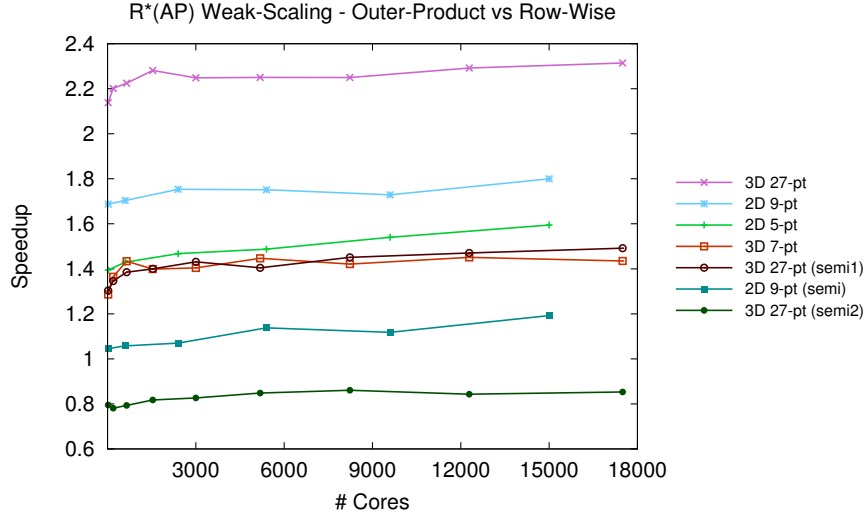


Figure 4. Speedups of the outer-product algorithm over the row-wise algorithm for model problems as observed on Edison. The number of fine grid points per core is approximately 100K for all problems.

Overall, the speedups are fairly constant as the core count increases, which implies that the two algorithms are scaling similarly. We observe that in some cases (particularly for the 2D or semi-coarsening problems at low core counts), the outer-product multiplication $R \cdot (AP)$ is slower than the row-wise multiplication; however, including the cost of the redistribution of R , the outer-product algorithm is a net improvement in all cases except 3D 27-point (semi2), as shown in Figure 4.

In Table 3, we present the ratios of per-processor communication costs for row-wise compared to outer-product algorithms for the $A_c = R \cdot (AP)$ matrix multiplication. The communication cost is the maximum over all processors of the sum of the amount of data sent and received. We note that the measured communication validates the theoretical analysis from Section 5.2, both in terms of the number of rows communicated as well as the total amount of data. The theoretical numbers in Table 3 assume perfect coarsening (that all local grid dimensions are multiples of 3) and ignore lower order terms (corresponding to edge and corner halo points); these effects make an impact for the 3D problems we consider. For example, in the 3D problems, edge and corner grid points constitute about 4% of each processor’s fine grid halo and about 11% of the coarse grid halo. In the cases of semicoarsening, the inaccuracy of the ratio of number of rows communicated is due to a slight inefficiency in our outer-product implementation; while we do not expect much of a performance improvement, we plan to update our implementation for use in future applications. Recall that the number of rows communicated during the row-wise algorithm corresponds to the processor’s fine grid halo (with respect to R in row-wise distribution), while that of the outer-product algorithm corresponds to the processor’s coarse grid halo

Fine Grid Operator	Rows		Data	
	Measured	Theory	Measured	Theory
2D 5-point	2.4	-	1.7	-
2D 9-point	3.0	3	2.3	2.3
2D 9-point (semi)	0.8	1.0	1.0	1.0
3D 7-point	4.0	-	2.2	-
3D 27-point	7.9	9.0	4.7	5.4
3D 27-point (semi1)	2.3	3.0	2.1	2.3
3D 27-point (semi2)	0.7	1.0	1.0	1.0
Nalu-Edge	2.3	-	2.3	-
Nalu-Element	7.9	-	3.7	-

Table 3. Factors by which the communication cost of the row-wise algorithm exceeds that of the outer-product algorithm for $A_c = R \cdot (AP)$ for model problems. The measured ratio is based on actual implementation while the theoretical ratio is the result of the analysis from Section 5.2. “Rows” corresponds to the number of rows sent and received and reflects the relative sizes of the maximum fine grid halo and the maximum coarse grid halo. “Data” corresponds to the maximum amount of data sent and received over all processors.

(with respect to R in column-wise distribution). The difference in ratios between rows and actual data demonstrates that the number of nonzeros per row communicated is greater for the outer-product algorithm than the row-wise algorithm, but overall the outer-product algorithm provides a net reduction.

To demonstrate more clearly the differences in the two approaches to computing $R \cdot (AP)$, we present a time breakdown plot in Figure 5 for the 3D 27-point problem at various scales. Recall that the row-wise approach requires a redistribution of R , in this case an explicit transpose of P . The transpose operation consists of a transpose of local data (“Local Transpose”), a communication phase to achieve row-wise distribution (“Communication (TransP)”), and then a merge phase to unify the local matrix data structure (“Local Merge (TransP)”). The row-wise algorithm for $R \cdot (AP)$ consists of a communication phase (“Communication (RAP)”) followed by the actual multiplication (“Local Multiply”). The outer-product algorithm consists of a local transpose of P to obtain R in column-wise distribution (“Local Transpose”), the matrix multiplication (“Local Multiply”), a communication phase to achieve row-wise distribution of A_c (“Communication (RAP)”), and finally a merge phase to finish reducing the final result (“Local Merge (RAP)”). The communication costs include the time to pack and unpack messages into buffers and convert between local and global information, all of which is proportional to the amount of data being communicated.

Note that both approaches share the local transpose and local multiply, shown at the bottom of the plots, and we expect those costs to be nearly equal. The two main benefits

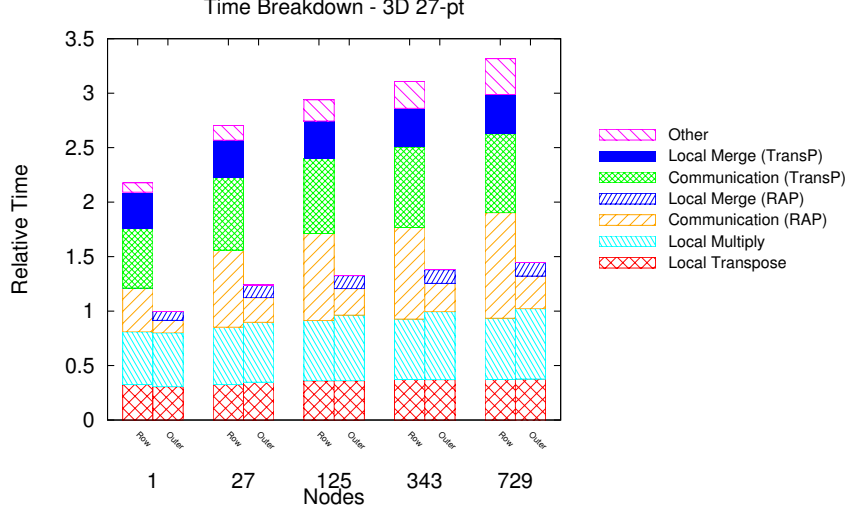
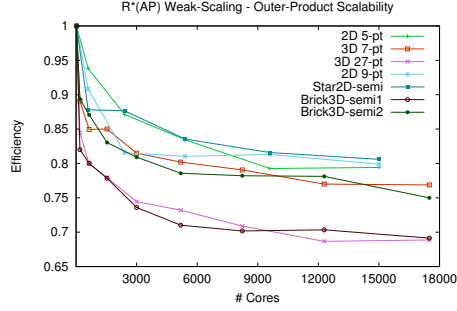


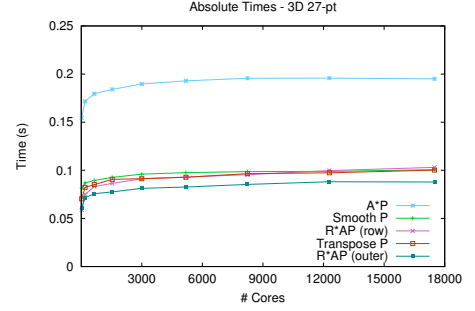
Figure 5. Time breakdown of the two approaches for $R \cdot (AP)$ for various numbers of nodes. The row-wise approach (on the left of each pair) includes both redistribution of R (by explicitly transposing P) and the matrix multiplication. The outer-product approach (on the right of each pair) includes a transpose of the local entries of P to obtain R in column-wise distribution as well the matrix multiplication. Communication includes the cost of packing and unpacking message buffers. All values have been normalized to the cost of the outer-product algorithm on 1 node.

of the outer-product approach are the reduction in communication during the $R \cdot (AP)$ operation and the avoidance of the redistribution of R . The reduction in communication can be seen by comparing the “Communication (RAP)”; in the case of 729 nodes, the ratio is a factor of 3.3. The cost of the redistribution of R can be seen in the other two contributions with the “TransP” label. The extra overhead of the outer-product algorithm is the final merge, given by “Local Merge (RAP),” which is relatively small.

For context, we provide two more plots for the model problem data. Figure 6(a) shows the weak-scaling of the outer-product algorithm for all model problems. Note that after the initial drop in performance from 1 node, the scaling is reasonable for all problems. While the 1 node experiment still involves 24 MPI processes, most processes own boundary regions of the domain and therefore communicate with fewer than the maximum number of nearest neighbors (in the 3D case, the 24 processes are arranged in a $2 \times 3 \times 4$ grid, so all processors are on the boundary). Thus, both cheaper intranode communication and boundary effects cause greater differences at the left end of the plot. Figure 6(b) shows the raw times for all sparse matrix multiplications involved in computing A_c (as well as the cost of explicit redistribution of P as required by the row-wise approach) for the 3D 27-point problem. Note



(a) Weak-scaling efficiency of the outer-product algorithm for $R \cdot (AP)$ for model problems.



(b) Absolute times for key operations in computing triple product for the 3D 27-point problem.

Figure 6. These plots provide context for the evaluation of the row-wise and outer-product approaches for the $R \cdot (AP)$ multiplication.

that the $A \cdot P$ multiplication has about twice the cost of the other operations even though its theoretical communication requirements are lower than the row-wise algorithm for $R \cdot (AP)$; for this problem, the $A \cdot P$ spends about 85% of its time in the local multiplication (likely bottlenecked by memory bandwidth). Note also that the cost of the outer-product algorithm for $R \cdot (AP)$ is less than the cost of explicitly transposing P (redistributing R) alone, and the $R \cdot (AP)$ operation even includes the cost of the transposing the local entries of P to obtain R in column-wise distribution.

6.3 Unstructured Problems

In this section we consider more realistic problems with unstructured grids arising from a fluid-flow application. In particular, we base our experiments on the SIERRA low Mach module/Nalu code, an unstructured, low Mach number variable density turbulent flow application code (see [20] for more details). The application code supports two types of discretizations: edge based (with connectivity similar to a 7-point stencil) and element-based (with connectivity similar to a 27-point stencil). The turbulence models used are in the class of modeling known as Large Eddy Simulations. We perform weak-scaling experiments, considering four levels of discretization of the problem running on 2, 16, 128, and 1024 nodes on Sky Bridge (16 cores per node).

As in the case of model problems, using the outer-product algorithm reduced communication compared to the row-wise algorithm for $R \cdot (AP)$. In the largest edge discretization, the row-wise algorithm communicates $2.3\times$ as many rows and $2.3\times$ as many total bytes. In the largest element discretization, those ratios are $7.9\times$ (for rows) and $3.7\times$ (for data). Note that the edge discretization involves connections among degrees of freedom of the unstruc-

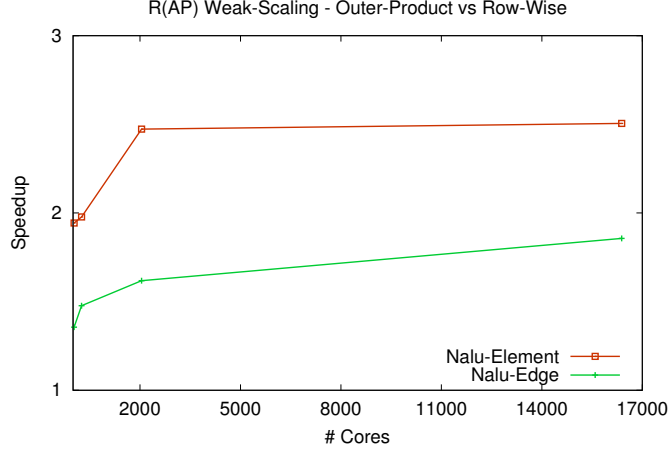


Figure 7. Speedups of the outer-product algorithm over the row-wise algorithm for fluid-flow problems as observed on Sky Bridge. The number of fine grid points per core is approximately 70K for all problems.

tured grid similar to a 7-point stencil on a structured grid, and the element discretization is more similar to a 27-point stencil. The communication ratios reflect this similarity (see Table 3).

Figure 7 shows the relative performance of the outer-product algorithm for $R \cdot (AP)$ as compared to the row-wise approach, which includes the cost of the redistribution of R . The largest speedup of $2.5\times$ is attained for the element discretization problem running on 1024 nodes (16,384 cores).

In Figure 8 we show the time breakdown of the $R \cdot (AP)$ multiplication for the element discretization. This plot matches that of Figure 5 for the model 3D 27-point problem. In this case, the reduction in communication for the actual multiplication is $2.2\times$ on 1024 nodes (note that the reduction in maximum data communicated by any processor is a factor of 3.7). As in the case of the 3D 27-point problem, we see that the reduced communication cost of the matrix multiplication as well as the avoidance of the explicit redistribution of P contribute to the overall speedup, despite the fact the the outer-product algorithm requires a slightly more expensive local matrix multiplication and a local merge step.

We present absolute times for all of the key operations in computing the triple product for the element discretization in Figure 9. As observed for the model problems, the $A \cdot P$ multiplication has the largest cost and is bottlenecked by local computation. Because that operation requires less communication than $R \cdot (AP)$ (using the row-wise algorithm), we expect that in more communication-bound situations, the cost of the second multiply will be slower and the benefit from the outer-product algorithm will be more pronounced.

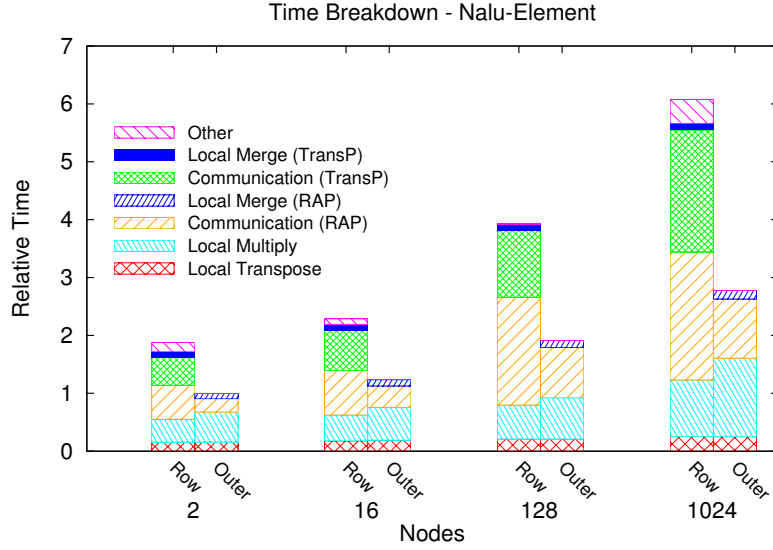


Figure 8. Time breakdown of the two approaches for $R \cdot (AP)$ for various numbers of nodes for the fluid flow problem with element discretization. See the caption of Figure 5 for more details of the plot.

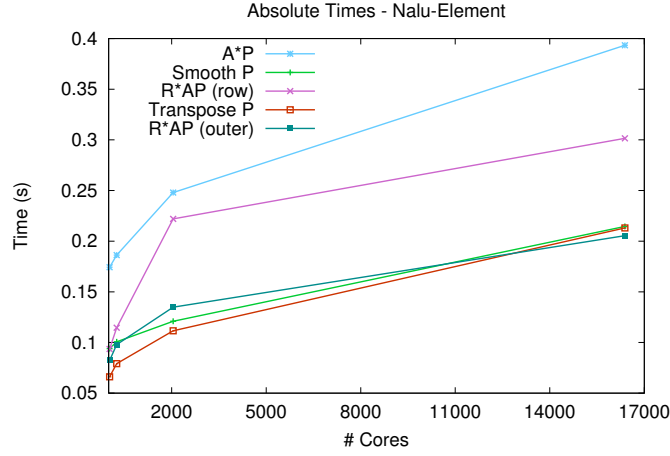


Figure 9. Absolute times for the key operations in computing the Galerkin triple product for the fluid-flow problem with element discretization.

7 Conclusions

In this paper, we consider the three SpMM operations within the Galerkin triple product of smoothed aggregation AMG. Our overall conclusion is that the row-wise algorithm is the best 1D method for the first two multiplications and that the outer-product algorithm is the best 1D method for the third multiplication. This conclusion comes from theoretical analysis applied to a model problem involving structured grids and a stencil operator, and it is supported by an empirical evaluation that focuses on the comparison between the row-wise and outer-product algorithms for the last SpMM.

While the exact communication costs of these SpMM operations are subject to the sparsity structures and parallel distributions of the matrices involved (and thus depend on the particular distribution and aggregation schemes used and their effects at processor boundaries), we found that our theoretical predictions for relative communication costs in the ideal case matched the empirical observations rather closely (see Table 3). Likewise, the unstructured problems with matrix sparsity that mimicked the model problems demonstrated similar communication behavior. Thus, we believe our conclusions that the outer-product algorithm effectively reduces communication are representative of many physical problems.

In particular, we identify the main reason that the outer-product algorithm achieves a communication reduction: the number of rows communicated is the size of the processor’s *coarse* grid halo instead of the *fine* grid halo. In nearly all cases, a processor’s coarse grid halo will be considerably smaller than the fine grid halo. However, the density of rows of the coarse grid also plays a role in the amount of communication required by the outer-product algorithm; this effect (and its relation to the effect on the row-wise algorithm) will be more problem dependent.

In our theoretical analysis, we ignored the costs of local computation and arrived at our algorithms of choice solely by their communication costs. Our empirical results show that, at least for these problems, the local computation requires a considerable amount of time, and even a large reduction in communication translates to a more modest decrease in total run time (see Figure 5, for example). We expect that the outer-product algorithm will have a more dramatic effect in more communication-bound situations, such as lower in the multigrid hierarchy and in strong-scaling regimes where the initial problem is small relative to the available number of processors.

Finally, we restricted our attention to performing the triple product with two multiplications and using only 1D SpMM algorithms. We argue in Section 5.4 that performing the triple product all-at-once increases the communication cost, but the reduced number of halo exchanges may prove to be net-beneficial for some problems, particularly when the number of neighbors is large (deep in the hierarchy). We do not consider 2D (or 3D – see [4]) SpMM algorithms in this work, and because those sets of algorithms include all 1D ones, they can only reduce communication further. We chose not to consider them primarily because they would involve more fundamental changes to our underlying software framework, but also because we believe they will not offer significant advantage at the highest levels of the hierarchy.

(which we observed to be the greatest bottleneck). We plan to investigate these alternatives in future work.

Acknowledgments

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This research was supported in part by an appointment to the Sandia National Laboratories Truman Fellowship in National Security Science and Engineering, sponsored by Sandia Corporation (a wholly owned subsidiary of Lockheed Martin Corporation) as Operator of Sandia National Laboratories under its U.S. Department of Energy Contract No. DE-AC04-94AL85000.

References

- [1] K. Akbudak and C. Aykanat. Simultaneous input and output matrix partitioning for outer-product-parallel sparse matrix-matrix multiplication. *SIAM Journal on Scientific Computing*, 36(5):C568–C590, 2014.
- [2] C. G. Baker and M. A. Heroux. Tpetra, and the use of generic programming in scientific computing. *Scientific Programming*, 20(2):115–128, Apr. 2012.
- [3] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc Web page. <http://www.mcs.anl.gov/petsc>, 2014.
- [4] G. Ballard, A. Buluç, J. Demmel, L. Grigori, B. Lipshitz, O. Schwartz, and S. Toledo. Communication optimal parallel multiplication of sparse random matrices. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA ’13, pages 222–231. ACM, 2013.
- [5] G. Ballard, A. Druinsky, N. Knight, and O. Schwartz. Brief announcement: Hypergraph partitioning for parallel sparse matrix-matrix multiplication. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA ’15, pages 86–88. ACM, 2015.
- [6] N. Bell, S. Dalton, and L. Olson. Exposing fine-grained parallelism in algebraic multigrid methods. *SIAM Journal on Scientific Computing*, 34(4):C123–C152, 2012.
- [7] U. Borštnik, J. VandeVondele, V. Weber, and J. Hutter. Sparse matrix multiplication: The distributed block-compressed sparse row library. *Parallel Computing*, 40(5 - 6):47 – 58, 2014.
- [8] A. Brandt and O. E. Livne. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*. SIAM, 2011.
- [9] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, 2nd edition, 2000.
- [10] A. Buluç and J. R. Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal on Scientific Computing*, 34(4):170 – 191, 2012.
- [11] L. Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University, Bozeman, MN, 1969.
- [12] M. Challacombe. A general parallel sparse-blocked matrix multiply for linear scaling SCF theory. *Computer Physics Communications*, 128(1-2):93–107, 2000.
- [13] S. Dalton, N. Bell, and L. N. Olson. Optimizing sparse matrix-matrix multiplication for the GPU. Technical report, NVIDIA, 2013.

- [14] R. Falgout, T. Kolev, U. M. Yang, J. Schroder, and P. Vassilevski. hypre web page. http://computation.llnl.gov/project/linear_solvers/software.php, 2015.
- [15] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala. ML 5.0 smoothed aggregation user’s guide. Sandia National Laboratories, TR SAND2006-2649, 2006.
- [16] F. Gremse, A. Höfter, L. Schwen, F. Kiessling, and U. Naumann. GPU-accelerated sparse matrix-matrix multiplication by iterative row merging. *SIAM Journal on Scientific Computing*, 37(1):C54–C71, 2015.
- [17] W. Hackbusch. *Multigrid Methods and Applications*, volume 4 of *Computational Mathematics*. Springer–Verlag, Berlin, 1985.
- [18] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, 2005.
- [19] M. A. Heroux and J. M. Willenbring. A new overview of the Trilinos project. *Scientific Programming*, 20(2):83–88, 2012.
- [20] P. Lin, M. Bettencourt, S. Domino, T. Fisher, M. Hoemmen, J. Hu, E. Phipps, A. Prokopenko, S. Rajamanickam, C. Siefert, and S. Kennon. Towards extreme-scale simulations for low mach fluids with second-generation Trilinos. *Parallel Processing Letters*, 24(04):1442005, 2014.
- [21] M. McCourt, B. Smith, and H. Zhang. Sparse matrix-matrix products executed through coloring. *SIAM Journal on Matrix Analysis and Applications*, 36(1):90–109, 2015.
- [22] A. Prokopenko, J. J. Hu, T. A. Wiesner, C. M. Siefert, and R. S. Tuminaro. MueLu user’s guide 1.0. Technical Report SAND2014-18874, Sandia National Laboratories, 2014.
- [23] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid*. Academic Press, 2000.
- [24] R. S. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid : Aggregation strategies on massively parallel machines. *SC Conference*, 0:5, 2000.
- [25] R. A. van de Geijn and J. Watts. SUMMA: scalable universal matrix multiplication algorithm. *Concurrency - Practice and Experience*, 9(4):255–274, 1997.
- [26] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56:179–196, 1996.
- [27] U. Yang, 2015. Personal communication.
- [28] H. Zhang, 2015. Personal communication.

A Analysis for Table 2

Table 2 presents the communication cost for the three SpMM computations for each of three 1D algorithms, assuming the fine grid operator A is a 3^d -point stencil. The analysis for the first column (corresponding to the row-wise algorithm) is given in Section 5.2.1, and the analysis for the $A_c = R \cdot (AP)$ multiply using the outer-product algorithm is given in Section 5.2.2. Here we provide explanations for the other entries of the table.

The second column of Table 2 corresponds to the column-wise algorithm, where only entries of the left input matrix are communicated. The column-wise algorithm applied to $A \cdot \hat{P}$ requires no communication, assuming all matrices are column-wise distributed and aggregates do not cross processor boundaries. In this case, \hat{P} has no nonzeros at entries corresponding to local fine grid points and nonlocal coarse grid aggregates. In order to compute $P = A \cdot \hat{P}$ in column-wise distribution, each processor needs to receive columns of A corresponding to nonlocal fine grid points that are adjacent to local aggregates in the graph corresponding to \hat{P} . However, due to the properties of \hat{P} , this is the empty set.

Performing $AP = A \cdot P$ using the column-wise algorithm does require communication. Again, each processor needs to receive columns of A corresponding to nonlocal fine grid points that are adjacent to local aggregates in the graph corresponding to P . In this case, there are h_F such points, and each column of A consists of 3^d nonzeros. Thus, counting both sends and receives, the communication cost is $2 \cdot 3^d \cdot h_F$ for this algorithm.

If we use the column-wise algorithm for $A_c = R \cdot (AP)$, the communication cost is $2(5/3)^d \cdot h_F$. In this case, each processor needs to receive columns of R corresponding to nonlocal fine grid points that are adjacent to local coarse grid aggregates in the graph corresponding to AP . Since the graph corresponding to AP depends on pairs of 2-hop neighbors in the graph corresponding to A , the nonlocal fine points that are adjacent to local aggregates is a halo of width two. Thus, the number of columns needed by each processor is approximately $2h_F$, and each column of R has $(5/3)^d$ nonzeros. Again, sends and receives are balanced.

The third column of Table 2 corresponds to the outer-product algorithm, where only entries of the output matrix are communicated. The entries in the table assume the input matrices have matching column- and row-wise distributions, and the output is distributed row-wise. We first consider $P = A \cdot \hat{P}$. For a processor to compute its local rows of P , it needs access to all length-two paths from its local fine points to all aggregates that consist of one edge in the graph corresponding to A and one edge in the graph corresponding to \hat{P} . By the assumed distribution of A and \hat{P} , it can compute locally all paths that start from any fine point, visit a local fine point, and end at any aggregate. It must receive from other processors data corresponding to paths starting from a local fine point, visiting a nonlocal fine point, and ending at any aggregate. Such paths must start from an inner halo fine grid point, so the number of rows read is h_F . The number of aggregates reachable from an inner halo point, provided that the hop in the fine grid is taken to an outer halo point, is $(5/3)^{d-1}$ on average. Thus, the communication cost of the outer-product algorithm for computing

$P = A \cdot \hat{P}$ is $2 \cdot (5/3)^{d-1} \cdot h_F$. We note that the outer-product algorithm can compute P in column-wise distribution with no communication; the cost of explicitly redistributing P from column-wise to row-wise distribution is also $2 \cdot (5/3)^{d-1} \cdot h_F$.

Finally, we consider using the outer-product algorithm for $AP = A \cdot P$. In this case, each processor needs information corresponding to length-three paths that start at local fine points, take two hops in the graph corresponding to A , and end at an aggregate with the last hop in the graph corresponding to \hat{P} . In this case, a processor must receive data for paths that take their first hop from an inner halo fine point to an outer halo fine point. Here, the number of rows is again h_F , and the number of nonzeros in the (unreduced) rows corresponds to the number of aggregates reachable from a halo point under this path restriction, which is $2(7/3)^{d-1}$ on average. We note that the outer-product algorithm can compute AP in column-wise distribution for a cost of $7^{d-1} \cdot h_F$.

DISTRIBUTION:

1	MS 9159	Grey Ballard, 8966
1	MS 1318	Robert Hoekstra, 1426
1	MS 9159	Jonathan Hu, 1426
1	MS 1322	Christopher Siefert, 1443
1	MS 0899	Technical Library, 8944 (electronic copy)
1	MS 0359	D. Chavez, LDRD Office, 1911

